

Репликация между SQL- и NoSQL- базами данных: туда и обратно

Горякин Александр,
Разработчик высоконагруженных систем хранения данных,
Tarantool / VK



HighLoad++
2022

Яндекс



Содержание

Содержание

Содержание

Содержание

Содержание

01

Постановка задачи

Постановка задачи

Необходимо переливать данные между разными источниками и потребителями – как SQL, так и NoSQL.





Постановка задачи

Необходимо переливать данные между разными источниками и потребителями – как SQL, так и NoSQL.

Требования:

- Низкий лаг репликации (40-50 мс)





Постановка задачи

Необходимо переливать данные между разными источниками и потребителями – как SQL, так и NoSQL.

Требования:

- Низкий лаг репликации (40-50 мс)
- Низкая нагрузка на БД источник





Постановка задачи

Необходимо переливать данные между разными источниками и потребителями – как SQL, так и NoSQL.

Требования:

- Низкий лаг репликации (40-50 мс)
- Низкая нагрузка на БД источник
- Устойчивость к высоким нагрузкам





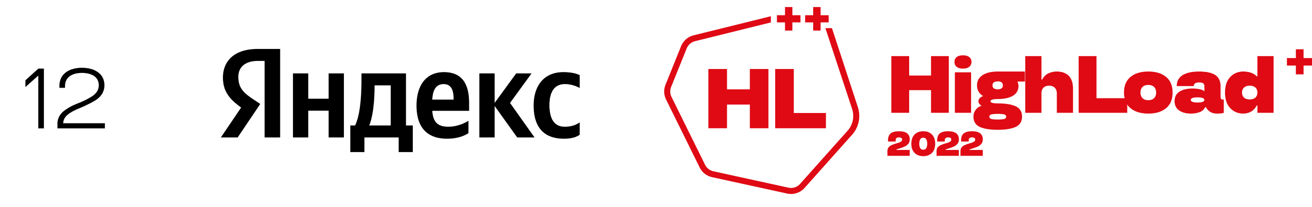
Постановка задачи

Необходимо переливать данные между разными источниками и потребителями – как SQL, так и NoSQL.

Требования:

- Низкий лаг репликации (40-50 мс)
- Низкая нагрузка на БД источник
- Устойчивость к высоким нагрузкам
- Отказоустойчивость при сбоях



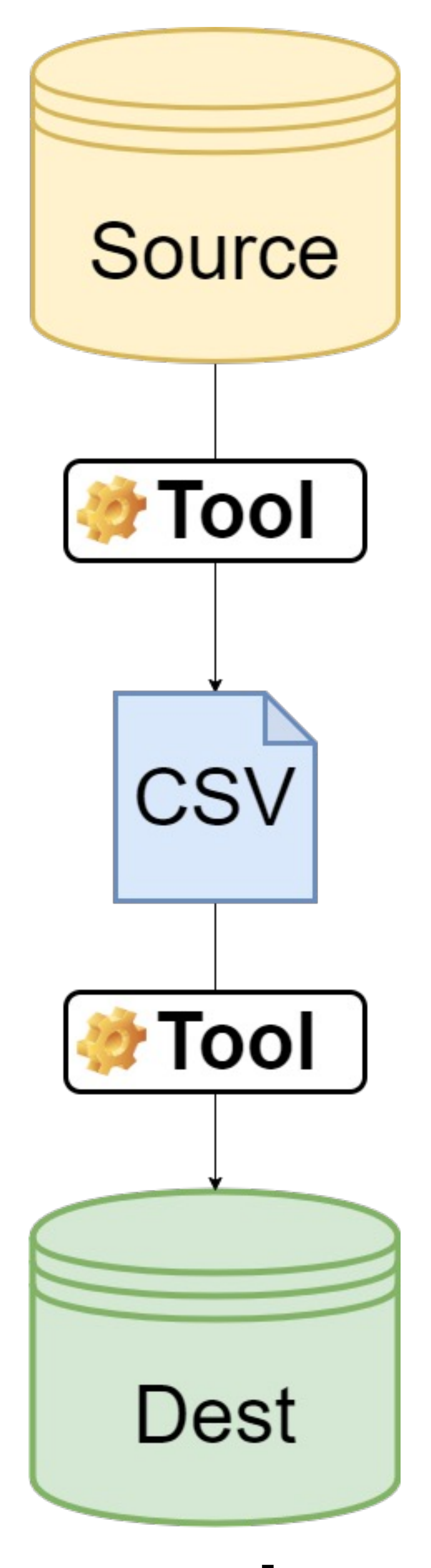


02

Варианты интеграции БД (ETL, CT, CDC)

Extract Transform Load
Change Tracking
Change Data Capture

ETL – Extract Transform Load



Плюсы:

- Универсальный механизм
- Человекочитаемый формат данных

Минусы:

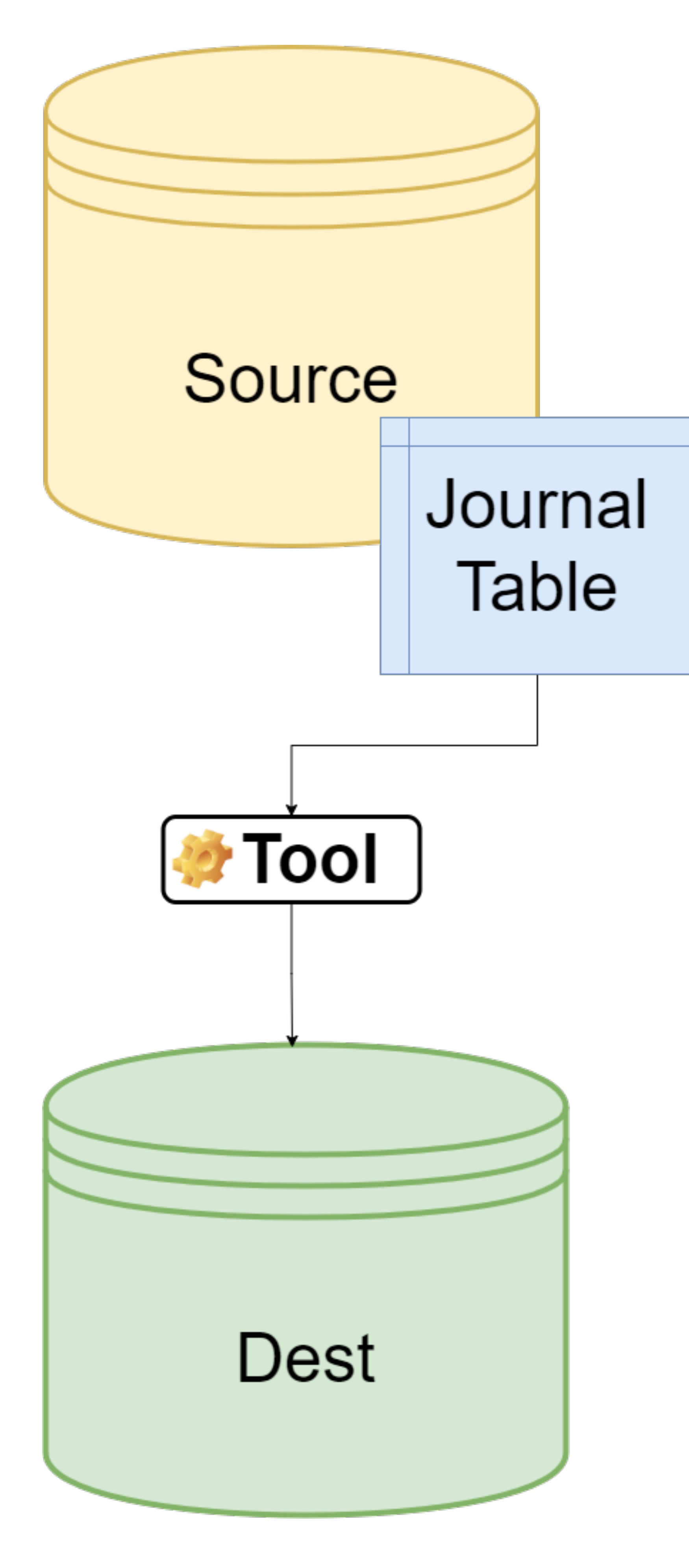
- Дополнительная нагрузка на источник и приемник
- Низкая скорость
- Огромный лаг
- Большой объем диска для хранения выгруженных данных







Change Tracking



Плюсы:

- Стриминг из источника
- Высокая скорость
- Низкий лаг

Минусы:

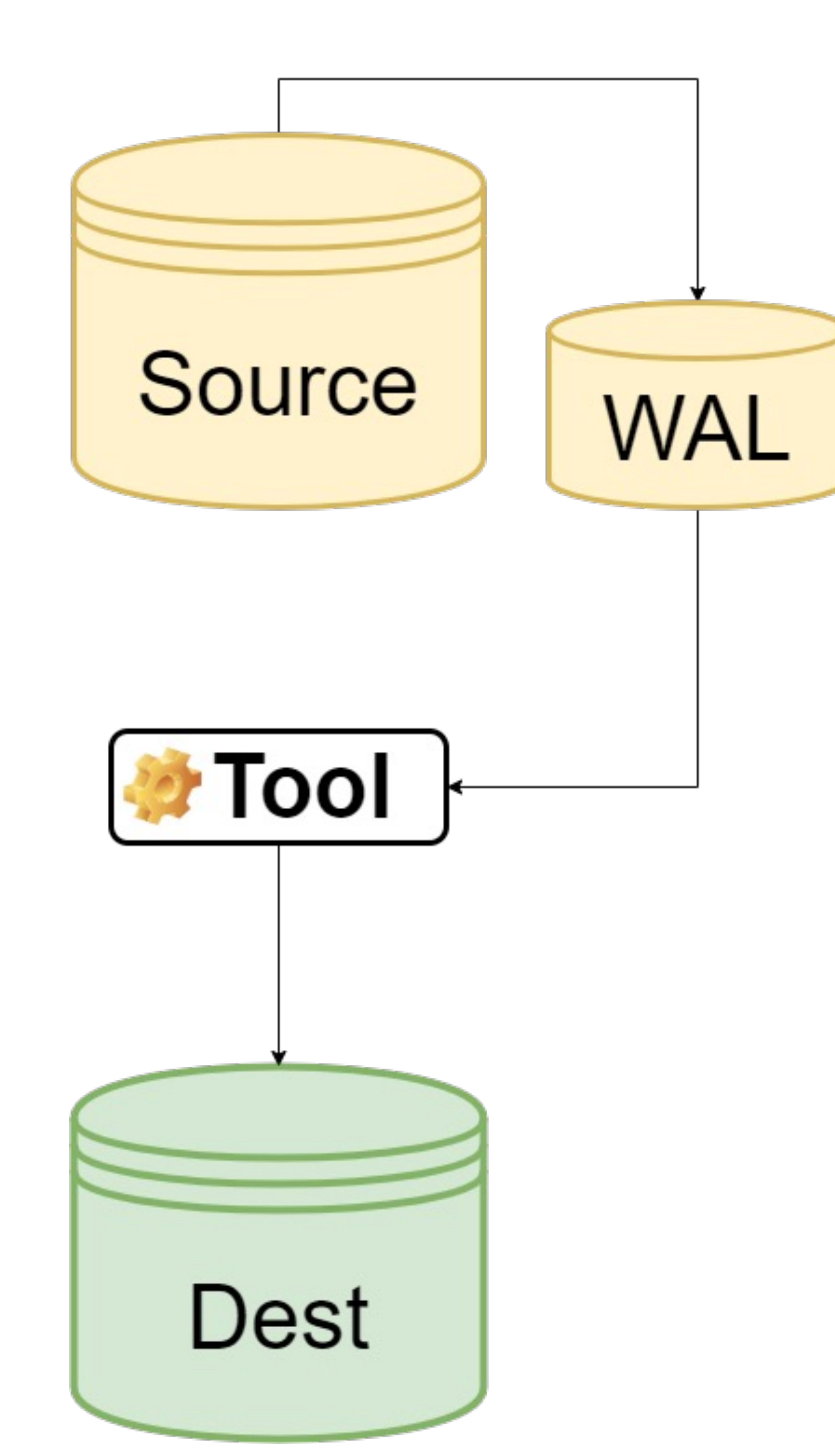
- Изменения в виде дельты
- Не все БД умеют из коробки
- Дополнительная нагрузка на источник и приемник
- Гарантии консистентности и тулинг свои для каждого кейса







CDC – Change Data Capture



Плюсы:

- Изменения приходят полностью
- Нет нагрузки на источник, использует WAL
- Высокая скорость
- Низкий лаг
- Гарантии консистентности данных

Минусы:

- Дополнительная нагрузка на приемник







03

Решения на основе CDC

Решения на основе CDC

- Oracle GoldenGate + Userexit

Плюсы:

- Быстро

Минусы:

- Сложно
- Дорого
- Только Oracle





Решения на основе CDC

- Oracle GoldenGate + Userexit
- IBM Infosphere CDC – GG для DB2

Плюсы:

- Быстро

Минусы:

- Сложно
- Дорого
- Только IBM DB2





Решения на основе CDC

- Oracle GoldenGate + Userexit
- IBM Infosphere CDC – GG для DB2
- Самописные (например, на основе pglogrepl + pgproto)

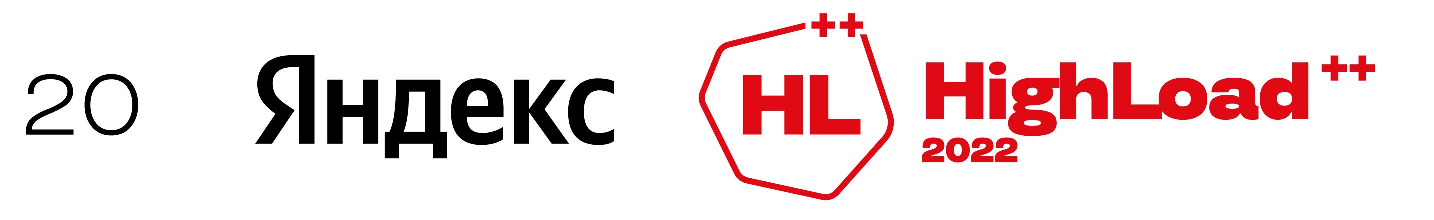
Плюсы:

- Понимаем, как работает
- Хорошо для конкретной задачи

Минусы:

- Долго разрабатывать
- Неуниверсальны
- Сложно поддерживать





Решения на основе CDC

- Oracle GoldenGate + Userexit
- IBM Infosphere CDC – GG для DB2
- Самописные (например, на основе pglogrepl + pgproto)
- Debezium





04

Debezium

Почему Debezium?
С чего мы начали
Debezium Embedded
Debezium Server

Почему Debezium?

- Написан на Java

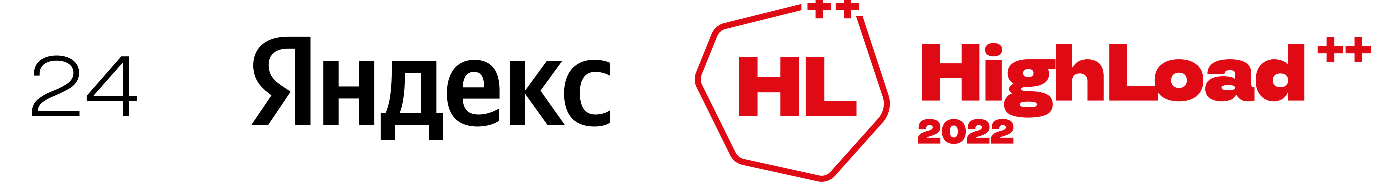




Почему Debezium?

- Написан на Java
- Основные компоненты
  - Source-коннекторы (к БД источникам)



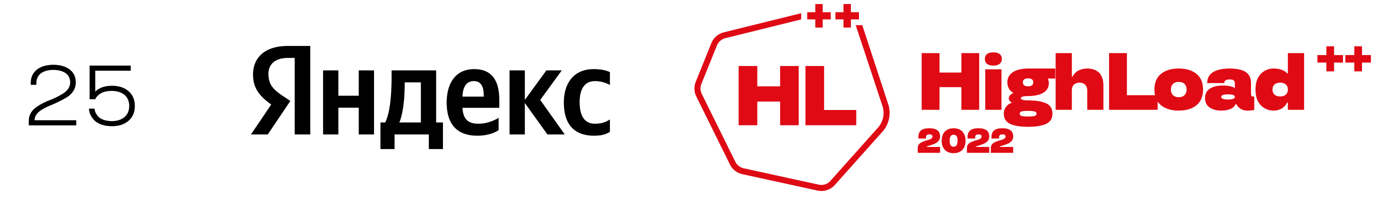




Почему Debezium?

- Написан на Java
- Основные компоненты
  - Source-коннекторы (к БД источникам)
  - Debezium Embedded (библиотека, без Kafka)





Почему Debezium?

- Написан на Java
- Основные компоненты
  - Source-коннекторы (к БД источникам)
  - Debezium Embedded (библиотека, без Kafka)
  - Debezium Server (готовое приложение, без Kafka)





Почему Debezium?

- Написан на Java
- Основные компоненты
  - Source-коннекторы (к БД источникам)
  - Debezium Embedded (библиотека, без Kafka)
  - Debezium Server (готовое приложение, без Kafka)
  - Sink-коннекторы (к целевым БД)





СПАКУХА ЩАС ВСЁ БУДЕТ

risovach.ru

С чего мы начали

Попытка с помощью Debezium заменить Oracle GoldenGate (Debezium Embedded)

- Чтение redo-логов с помощью LogMiner





С чего мы начали

Попытка с помощью Debezium заменить Oracle GoldenGate (Debezium Embedded)

- Чтение redo-логов с помощью LogMiner
- Чтение изменений с помощью XStream API





С чего мы начали

Попытка с помощью Debezium заменить Oracle GoldenGate (Debezium Embedded)

- Чтение redo-логов с помощью LogMiner
- Чтение изменений с помощью XStream API

Но:

- LogMiner нестабилен

С чего мы начали

Попытка с помощью Debezium заменить Oracle GoldenGate (Debezium Embedded)

- Чтение redo-логов с помощью LogMiner
- Чтение изменений с помощью XStream API

Но:

- LogMiner нестабилен
- LogMiner deprecated





С чего мы начали

Попытка с помощью Debezium заменить Oracle GoldenGate (Debezium Embedded)

- Чтение redo-логов с помощью LogMiner
- Чтение изменений с помощью XStream API

Но:

- LogMiner нестабилен
- LogMiner deprecated
- Спецификации LogMiner меняются от версии к версии

С чего мы начали

Попытка с помощью Debezium заменить Oracle GoldenGate (Debezium Embedded)

- Чтение redo-логов с помощью LogMiner
- Чтение изменений с помощью XStream API

Но:

- LogMiner нестабилен
- LogMiner deprecated
- Спецификации LogMiner меняются от версии к версии
- XStream API по лицензии Oracle GoldenGate





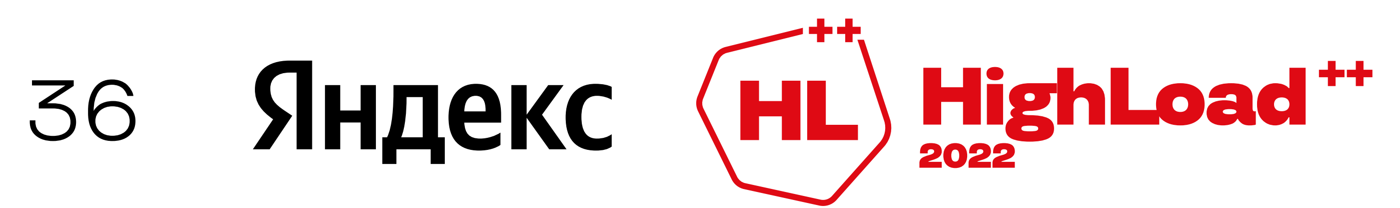


Debezium Embedded

Решили

- Попробовать на примере PostgreSQL





Debezium Embedded

Решили

- Попробовать на примере PostgreSQL
- Сделать более универсальным

Debezium Embedded

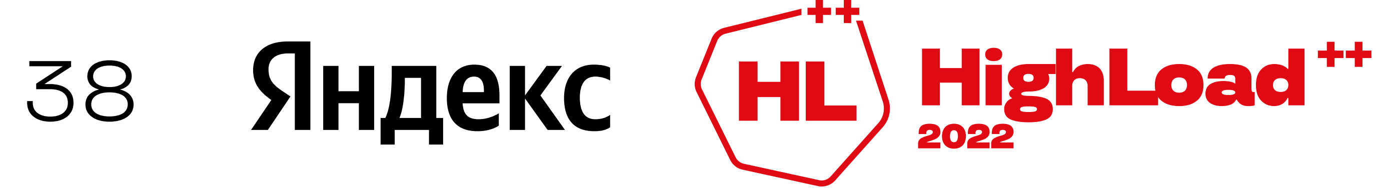
Решили

- Попробовать на примере PostgreSQL
- Сделать более универсальным

Получили

- Монолит





Debezium Embedded

Решили

- Попробовать на примере PostgreSQL
- Сделать более универсальным

Получили

- Монолит
- Высокий лаг репликации







Debezium Embedded

Решили

- Попробовать на примере PostgreSQL
- Сделать более универсальным

Получили

- Монолит
- Высокий лаг репликации
- Лаг репликации растет пропорционально количеству записей





Debezium Embedded

Решили

- Попробовать на примере PostgreSQL
- Сделать более универсальным

Получили

- Монолит
- Высокий лаг репликации
- Лаг репликации растет пропорционально количеству записей
- Записи приходят в JSON






Привет грабли!
Это снова, я!

Debezium Embedded – лаг

Попытались уменьшить лаг репликации

- Асинхронная прогрузка данных

Debezium Embedded – лаг

Попытались уменьшить лаг репликации

- Асинхронная прогрузка данных

Но

  - В целевой БД получаем неконсистентность данных





Debezium Embedded – лаг

Попытались уменьшить лаг репликации

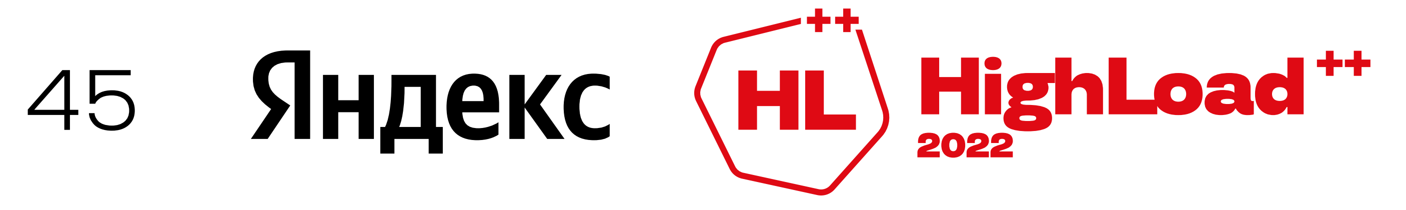
- Асинхронная прогрузка данных

Но

  - В целевой БД получаем неконсистентность данных

- Перешли от JSON к Kafka SourceRecord





Debezium Embedded – лаг

Попытались уменьшить лаг репликации

- Асинхронная прогрузка данных

Но

  - В целевой БД получаем неконсистентность данных

- Перешли от JSON к Kafka SourceRecord
  - Незначительное уменьшение лага репликации (несколько минут)





Debezium Embedded – лаг

Попытались уменьшить лаг репликации

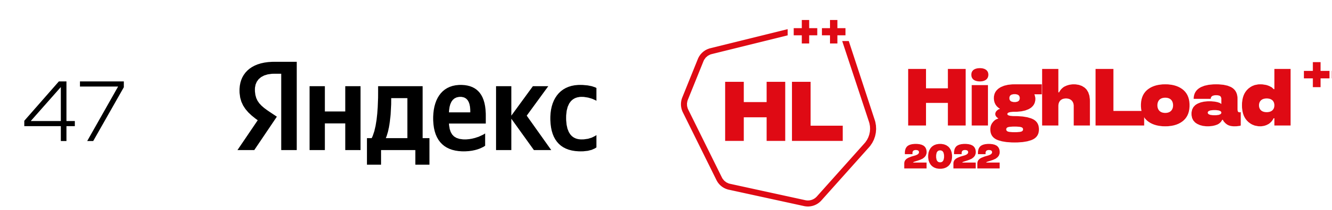
- Асинхронная прогрузка данных

Но

  - В целевой БД получаем неконсистентность данных

- Перешли от JSON к Kafka SourceRecord
  - Незначительное уменьшение лага репликации (несколько минут)
- Батчинг





Debezium Embedded – лаг

Попытались уменьшить лаг репликации

- Асинхронная прогрузка данных

Но

  - В целевой БД получаем неконсистентность данных
- Перешли от JSON к Kafka SourceRecord
  - Незначительное уменьшение лага репликации (несколько минут)
- Батчинг
  - Ещё немного уменьшили лаг (в 1.5-2 раза)





Debezium Embedded – лаг

Попытались уменьшить лаг репликации

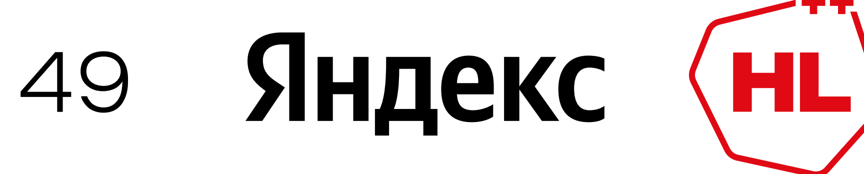
- Асинхронная прогрузка данных

Но

  - В целевой БД получаем неконсистентность данных

- Перешли от JSON к Kafka SourceRecord
  - Незначительное уменьшение лага репликации (несколько минут)
- Батчинг
  - Ещё немного уменьшили лаг (в 1.5-2 раза)
- Батчинг + отправка в отдельном потоке







Debezium Embedded – лаг

Попытались уменьшить лаг репликации

- Асинхронная прогрузка данных

Но

  - В целевой БД получаем неконсистентность данных

- Перешли от JSON к Kafka SourceRecord
  - Незначительное уменьшение лага репликации (несколько минут)
- Батчинг
  - Ещё немного уменьшили лаг (в 1.5-2 раза)
- Батчинг + отправка в отдельном потоке
  - Приемлемый лаг репликации (~40-50 мс)





Debezium Embedded – отказоустойчивость

Из коробки хранит оффсеты

- В файле (легко случайно удалить)





Debezium Embedded – отказоустойчивость

Из коробки хранит оффсеты

- В файле (легко случайно удалить)
- В Kafka (может быть недоступна)





Debezium Embedded – отказоустойчивость

Из коробки хранит оффсеты

- В файле (легко случайно удалить)
- В Kafka (может быть недоступна)

При падении может не восстановиться

- Храним оффсеты в БД-приёмнике





Debezium Embedded – отказоустойчивость

Из коробки хранит оффсеты

- В файле (легко случайно удалить)
- В Kafka (может быть недоступна)

При падении может не восстановиться

- Храним оффсеты в БД-приёмнике
- Читаем их при старте







Debezium Embedded – отказоустойчивость

Из коробки хранит оффсеты

- В файле (легко случайно удалить)
- В Kafka (может быть недоступна)

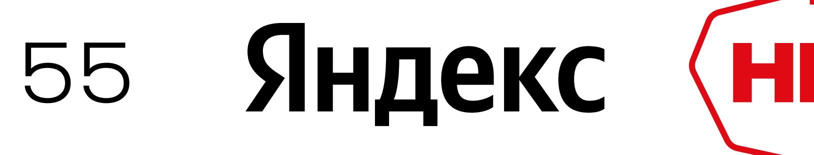
При падении может не восстановиться

- Храним оффсеты в БД-приёмнике
- Читаем их при старте

Ошибка записи в БД-приёмник

- Из коробки продолжит работать – плохо







Debezium Embedded – отказоустойчивость

Из коробки хранит оффсеты
- В файле (легко случайно удалить)
- В Kafka (может быть недоступна)

При падении может не восстановиться
- Храним оффсеты в БД-приёмнике
- Читаем их при старте

Ошибка записи в БД-приёмник
- Из коробки продолжит работать – плохо
- Делаем повторные попытки записать батч





Debezium Embedded – отказоустойчивость

Из коробки хранит оффсеты

- В файле (легко случайно удалить)
- В Kafka (может быть недоступна)

При падении может не восстановиться

- Храним оффсеты в БД-приёмнике
- Читаем их при старте

Ошибка записи в БД-приёмник

- Из коробки продолжит работать – плохо
- Делаем повторные попытки записать батч
- Начинаем отправку с ошибочной строки







Для чего хранить оффсеты?

SRC

Поток данных из источника (репликатор)

I U I I U I

Шард 1 Шард 2 Шард 3

Tarantool

Debezium Embedded – монолит

Т.к. приложение монолитное

Плюсы:

- Все коннекторы доступны из коробки





Debezium Embedded – монолит

Т.к. приложение монолитное

Плюсы:

- Все коннекторы доступны из коробки
- Относительно гибкая конфигурация





Debezium Embedded – монолит

Т.к. приложение монолитное

Плюсы:

- Все коннекторы доступны из коробки
- Относительно гибкая конфигурация

Минусы:

- Сложно масштабировать







Debezium Embedded – монолит

Т.к. приложение монолитное

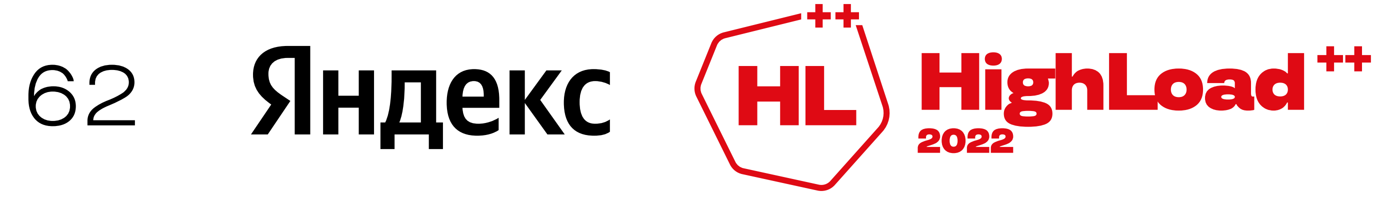
Плюсы:

- Все коннекторы доступны из коробки
- Относительно гибкая конфигурация

Минусы:

- Сложно масштабировать
- Любое изменение –> пересборка всего приложения





Debezium Server

MySQL

PostgreSQL

Debezium MySQL

Debezium PostgreSQL

Source Connectors

Debezium Server

redis

amazon KINESIS

PULSAR

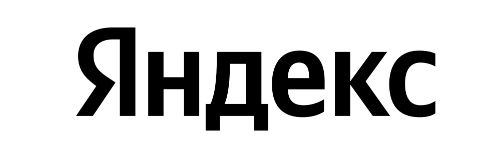
Google Pub/Sub

Debezium Server

Плюсы:

- Есть готовые сборки







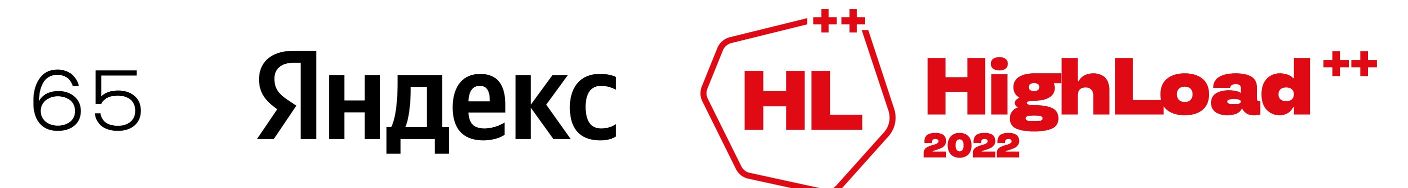
Debezium Server

Плюсы:

- Есть готовые сборки
- Легко добавить новый source-коннектор







Debezium Server

Плюсы:

- Есть готовые сборки
- Легко добавить новый source-коннектор
- Легко использовать с собственными source-коннекторами



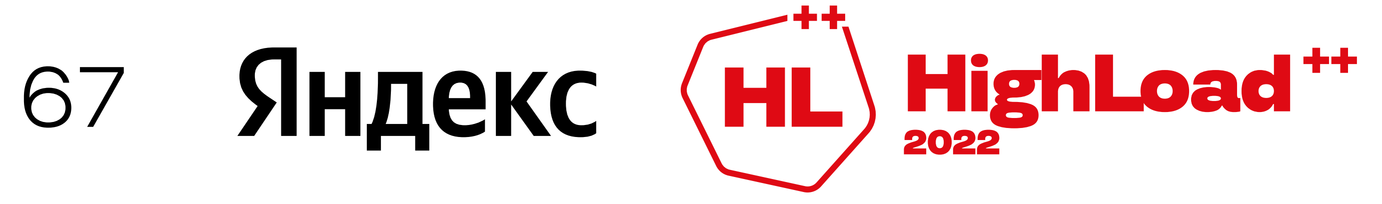


Debezium Server

Плюсы:

- Есть готовые сборки
- Легко добавить новый source-коннектор
- Легко использовать с собственными source-коннекторами
- Просто конфигурируется





Debezium Server

Плюсы:

- Есть готовые сборки
- Легко добавить новый source-коннектор
- Легко использовать с собственными source-коннекторами
- Просто конфигурируется
- Легко масштабируется с помощью Kubernetes



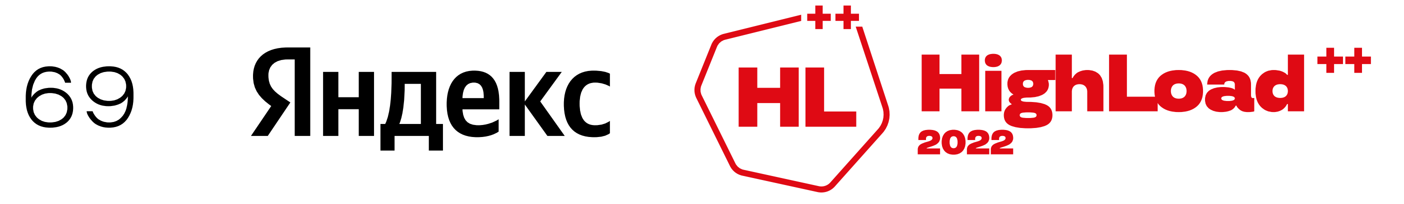


Debezium Server

Плюсы:

- Есть готовые сборки
- Легко добавить новый source-коннектор
- Легко использовать с собственными source-коннекторами
- Просто конфигурируется
- Легко масштабируется с помощью Kubernetes
- Можно добавить своё хранилище оффсетов





Debezium Server

Плюсы:

- Есть готовые сборки
- Легко добавить новый source-коннектор
- Легко использовать с собственными source-коннекторами
- Просто конфигурируется
- Легко масштабируется с помощью Kubernetes
- Можно добавить своё хранилище оффсетов

Минусы:

- Сложность добавления своего sink-коннектора (относительная)







Debezium Server

Плюсы:

- Есть готовые сборки
- Легко добавить новый source-коннектор
- Легко использовать с собственными source-коннекторами
- Просто конфигурируется
- Легко масштабируется с помощью Kubernetes
- Можно добавить своё хранилище оффсетов

Минусы:

- Сложность добавления своего sink-коннектора (относительная)
- На каждое изменение sink'а надо пересобирать дистрибутив







Debezium Server

Плюсы:

- Есть готовые сборки
- Легко добавить новый source-коннектор
- Легко использовать с собственными source-коннекторами
- Просто конфигурируется
- Легко масштабируется с помощью Kubernetes
- Можно добавить своё хранилище оффсетов

Минусы:

- Сложность добавления своего sink-коннектора (относительная)
- На каждое изменение sink'а надо пересобирать дистрибутив
- Из коробки хранит оффсеты в файле, в Kafka или в Redis







А если источник распределенный?

А что делать если источник шардирован, как, например, MongoDB или Tarantool?

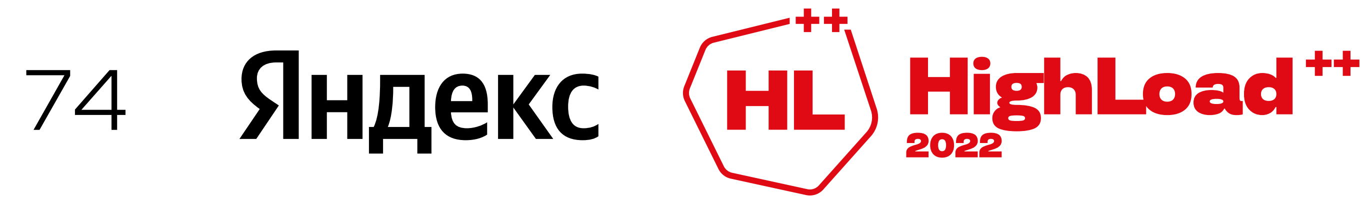
А если источник распределенный?

А что делать если источник шардирован, как, например, MongoDB или Tarantool?

Debezium Embedded

- Масштабируем по шардам вручную





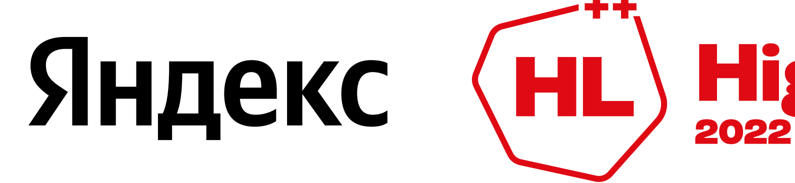
А если источник распределенный?

А что делать если источник шардирован, как, например, MongoDB или Tarantool?

Debezium Embedded

- Масштабируем по шардам вручную
- Оффсеты храним в приемнике для каждого шарда





А если источник распределенный?

А что делать если источник шардирован, как, например, MongoDB или Tarantool?

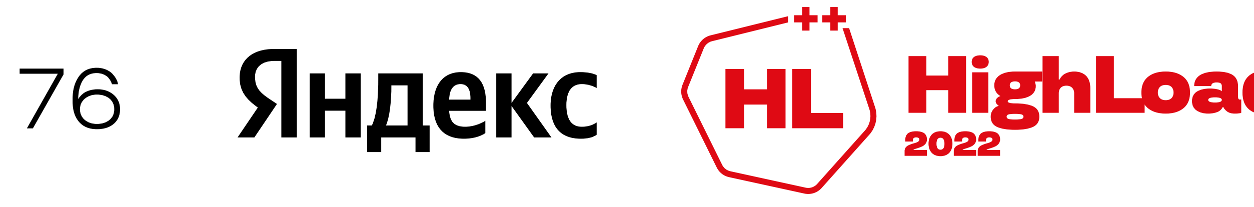
Debezium Embedded

- Масштабируем по шардам вручную
- Оффсеты храним в приемнике для каждого шарда

Debezium Server

- Масштабируем по шардам с помощью Kubernetes





А если источник распределенный?

А что делать если источник шардирован, как, например, MongoDB или Tarantool?

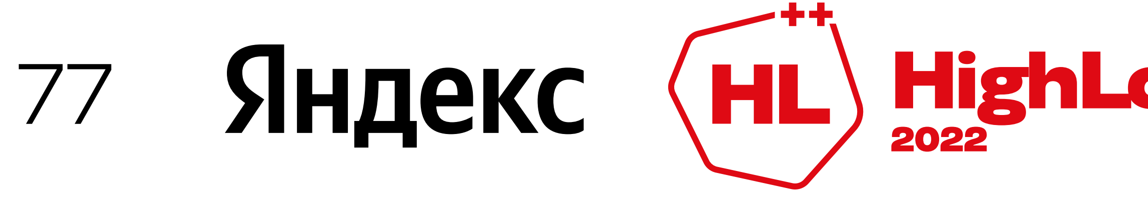
Debezium Embedded

- Масштабируем по шардам вручную
- Оффсеты храним в приемнике для каждого шарда

Debezium Server

- Масштабируем по шардам с помощью Kubernetes
- Оффсеты храним в файле, Kafka, Redis, приемнике для каждого шарда





А если источник распределенный?

А что делать если источник шардирован, как, например, MongoDB или Tarantool?

Debezium Embedded

- Масштабируем по шардам вручную
- Оффсеты храним в приемнике для каждого шарда

Debezium Server

- Масштабируем по шардам с помощью Kubernetes
- Оффсеты храним в файле, Kafka, Redis, приемнике для каждого шарда

Kafka-connect

- Масштабируем по шардам с помощью тасков kafka-connect







А если источник распределенный?

А что делать если источник шардирован, как, например, MongoDB или Tarantool?

Debezium Embedded

- Масштабируем по шардам вручную
- Оффсеты храним в приемнике для каждого шарда

Debezium Server

- Масштабируем по шардам с помощью Kubernetes
- Оффсеты храним в файле, Kafka, Redis, приемнике для каждого шарда

Kafka-connect

- Масштабируем по шардам с помощью тасков kafka-connect
- Храним оффсеты в Kafka для каждого шарда







05

Выводы

Выводы

- Выбрали CDC

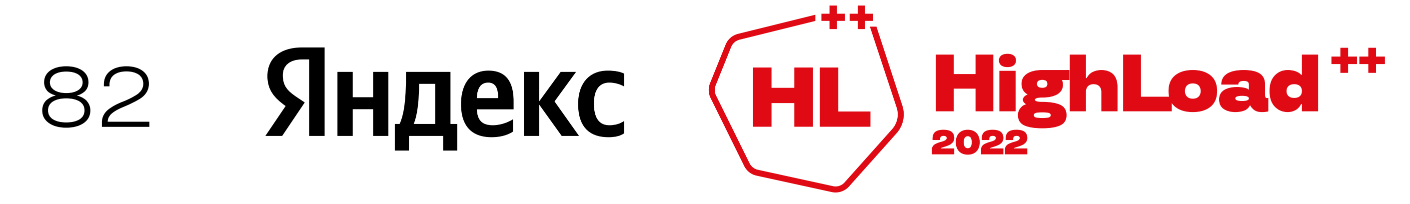




Выводы

- Выбрали CDC
- Java + Debezium





Выводы

- Выбрали CDC
- Java + Debezium
- Собрали кейсы неправильного использования





Выводы

- Выбрали CDC
- Java + Debezium
- Собрали кейсы неправильного использования
- Получили инструмент, реализующий необходимый функционал

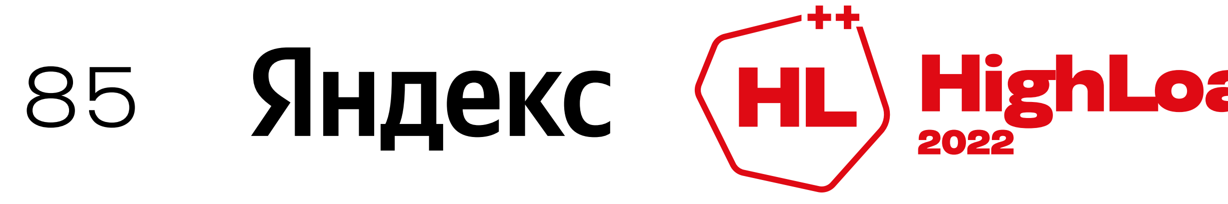




Выводы

- Выбрали CDC
- Java + Debezium
- Собрали кейсы неправильного использования
- Получили инструмент, реализующий необходимый функционал
- Debezium Embedded –> Debezium Server





Спасибо!

Александр Горякин

Разработчик высоконагруженных СХД,
Tarantool / VK

al.goryakin@corp.mail.ru
@droidroot1995



